Red Hat Trusted Artifact Signer 1

Deployment Guide

Installing and configuring the Trusted Artifact Signer service on Red Hat OpenShift

Last Updated: 2024-06-27

Red Hat Trusted Artifact Signer 1 Deployment Guide

Installing and configuring the Trusted Artifact Signer service on Red Hat OpenShift

Abstract

This Deployment Guide gives you guidance on installing the Trusted Artifact Signer service on Red Hat OpenShift, and verifying that installation was successful. Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message

Table of Contents

PREFACE

Welcome to the Red Hat Trusted Artifact Signer Deployment Guide!

These procedures can help guide you on deploying the full Trusted Artifact Signer (RHTAS) software stack on OpenShift, and verify that deployment. You can view the official RHTAS Release Notes here.

CHAPTER 1. INSTALLING TRUSTED ARTIFACT SIGNER USING THE OPERATOR LIFECYCLE MANAGER

You can install the Red Hat Trusted Artifact Signer (RHTAS) operator, and deploy the RHTAS service by using OpenShift's Operator Lifecycle Manager (OLM). This deployment gives you a basic signing framework with your choice of an OpenID Connect (OIDC) provider. You must configure at least one of the following OIDC providers: Red Hat Single Sign-on (SSO), Google, Amazon Secure Token Service (STS), or GitHub. You can also optionally customize your database solution, if you do not want to use the default.

Prerequisites

- Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.
- Access to the OpenShift web console with the **cluster-admin** role.
- A workstation with the **oc** binary installed.

Procedure

1. Log in to the OpenShift web console with a user that has the **cluster-admin** role.
2. From the **Administrator** perspective, expand the **Operators** navigation menu, and click **OperatorHub**.
3. In the search field, type **trusted**, and click the **Red Hat Trusted Artifact Signer** tile.
4. Click the **Install** button to show the operator details.
5. Accept the default values, click **Install** on the *Install Operator* page, and wait for the installation to finish.

   NOTE

   The Trusted Artifact Signer operator installs into the **openshift-operators** namespace, and all dependencies are automatically installed.

6. Once the installation finishes, a new project is automatically create for you. The new project name is **trusted-artifact-signer**.
7. Optional. Instead of the default database, you can use an alternative database provider for the Trusted Artifact Signer service. If you want to use Amazon's Relational Database Service (RDS), or a self-managed database on OpenShift, then follow one of those procedures first before continuing on with this installation. Once done configuring one of these other database providers, you can continue onto the next step of this procedure.
8. To deploy the Trusted Artifact Signer service.
   a. Expand **Operators** from the navigation menu, click **Installed Operators**.
   b. Select **trusted-artifact-signer** from the project drop-down box.
   c. Click **Red Hat Trusted Artifact Signer**.
   d. Click the **Securesign** tab, and click the **Create Securesign** button.

e. On the *Create Securesign* page, select **YAML view**.

f. You can configure Google OAuth, Amazon STS, Red Hat's SSO, or GitHub OAuth as the initial OIDC provider during this deployment. Under the **spec.fulcio.config.OIDCIssuers** section, edit the following three lines with the OIDC provider URL, and set the **ClientID** appropriately.

Example

```
...
OIDCIssuers:
  - Issuer: 'OIDC_ISSUER_URL':
    ClientID: CLIENT_ID
    IssuerURL: 'OIDC_ISSUER_URL'
    Type: email
...
```

IMPORTANT

You can define several different OIDC providers in the same configuration.

NOTE

If Red Hat's SSO is already implemented as your OIDC provider, then run the following command to find the issuer URL:

```
$ echo https://$(oc get route keycloak -n keycloak-system | tail -n 1 | awk
'{print $2}')/auth/realms/trusted-artifact-signer
```

Set the **ClientID** to **trusted-artifact-signer**.

g. Optional. If using a different database other than the default, then under the **spec.trillian** section, set **create** to **false**, and give the name of the database secret object.

Example

```
...
trillian:
  database:
    create: false
    databaseSecretRef:
      name: trillian-mysql
...
```

h. Click the **Create** button.

9. Click **All instances** tab to watch the deployment status until the **CTlog**, **Fulcio**, **Rekor**, **Trillian**, and **TUF** instances are ready.

NOTE

The **Securesign** instance does not give a status.

10. You can check on the health of the new Trusted Artifact Signer service by using Prometheus in the OpenShift console. From the navigation menu, expand **Observe**, and click **Dashboards**.

11. Verify the installation by signing a container image, or a Git commit.

Additional resources

- See the Appendix in the RHTAS Deployment Guide for more information about RHTAS components and version numbers.

CHAPTER 2. VERIFY THE TRUSTED ARTIFACT SIGNER SERVICE INSTALLATION

2.1. SIGNING AND VERIFYING CONTAINERS BY USING COSIGN FROM THE COMMAND-LINE INTERFACE

The **cosign** tool gives you the capability to sign and verify Open Container Initiative (OCI) container images, along with other build artifacts by using Red Hat's Trusted Artifact Signer (RHTAS) service.

Prerequisites

- A RHTAS installation on Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.
- Access to the OpenShift web console.
- A workstation with the **podman** binary installed.

Procedure

1. Download the **cosign** binary from the OpenShift cluster to your workstation.
    a. Login to the OpenShift web console. From the home page, click the **?** icon, click **Command line tools**, go to the **cosign** download section, and click the link for your platform.
    b. Open a terminal on your workstation, decompress the binary **.gz** file, and set the execute bit:

    Example

    ```
    $ gunzip cosign-amd64.gz
    $ chmod +x cosign-amd64
    ```

    c. Move and rename the binary to a location within your **$PATH** environment:

    Example

    ```
    $ sudo mv cosign-amd64 /usr/local/bin/cosign
    ```

2. Switch to the RHTAS project:

    Syntax

    ```
    oc project PROJECT_NAME
    ```

    Example

    ```
    $ oc project trusted-artifact-signer
    ```

    NOTE

    Use the project name for the RHTAS installation.

3. Configure your shell environment for doing container image signing and verifying.

   Example

   ```
   $ export TUF_URL=$(oc get tuf -o jsonpath='{.items[0].status.url}' -n trusted-artifact-signer)
   $ export OIDC_ISSUER_URL=https://$(oc get route keycloak -n keycloak-system | tail -n 1 |
   awk '{print $2}')/auth/realms/trusted-artifact-signer
   $ export COSIGN_FULCIO_URL=$(oc get fulcio -o jsonpath='{.items[0].status.url}' -n
   trusted-artifact-signer)
   $ export COSIGN_REKOR_URL=$(oc get rekor -o jsonpath='{.items[0].status.url}' -n trusted-
   artifact-signer)
   $ export COSIGN_MIRROR=$TUF_URL
   $ export COSIGN_ROOT=$TUF_URL/root.json
   $ export COSIGN_OIDC_CLIENT_ID="trusted-artifact-signer"
   $ export COSIGN_OIDC_ISSUER=$OIDC_ISSUER_URL
   $ export COSIGN_CERTIFICATE_OIDC_ISSUER=$OIDC_ISSUER_URL
   $ export COSIGN_YES="true"
   $ export SIGSTORE_FULCIO_URL=$COSIGN_FULCIO_URL
   $ export SIGSTORE_OIDC_ISSUER=$COSIGN_OIDC_ISSUER
   $ export SIGSTORE_REKOR_URL=$COSIGN_REKOR_URL
   $ export REKOR_REKOR_SERVER=$COSIGN_REKOR_URL
   ```

4. Initialize The Update Framework (TUF) system:

   Example

   ```
   $ cosign initialize
   ```

5. Sign a test container image.

   a. Create an empty container image:

      Example

      ```
      $ echo "FROM scratch" > ./tmp.Dockerfile
      $ podman build . -f ./tmp.Dockerfile -t ttl.sh/rhtas/test-image:1h
      ```

   b. Push the empty container image to the **ttl.sh** ephemeral registry:

      Example

      ```
      $ podman push ttl.sh/rhtas/test-image:1h
      ```

   c. Sign the container image:

      Syntax

      ```
      cosign sign -y IMAGE_NAME:TAG
      ```

      Example

      ```
      $ cosign sign -y ttl.sh/rhtas/test-image:1h
      ```

A web browser opens allowing you to sign the container image with an email address.

d. Remove the temporary Docker file:

Example

```
$ rm ./tmp.Dockerfile
```

6. Verify a signed container image by using a certificate identity and issuer:

Syntax

```
cosign verify --certificate-identity=SIGNING_EMAIL_ADDR IMAGE_NAME:TAG
```

Example

```
$ cosign verify --certificate-identity=jdoe@redhat.com ttl.sh/rhtas/test-image:1h
```

NOTE

You can also use regular expressions for the certificate identity and issuer by using the following options to the **cosign** command, **--certificate-identity-regexp** and **--certificate-oidc-issuer-regexp**.

Additional resources

- Installing Red Hat Trusted Artifact Signer on OpenShift .
- Customizing Red Hat Trusted Application Pipeline.
- See the Signing and verifying commits by using Gitsign from the command-line interface section of the RHTAS Deployment Guide for details on signing and verifying Git commits.
- The Update Framework home page.

2.2. SIGNING AND VERIFYING COMMITS BY USING GITSIGN FROM THE COMMAND-LINE INTERFACE

The **gitsign** tool gives you the ability to sign and verify Git repository commits by using Red Hat's Trusted Artifact Signer (RHTAS) service.

Prerequisites

- A RHTAS installation on Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.
- Access to the OpenShift web console.
- Download the **cosign** binary from the OpenShift cluster.

Procedure

1. Download the **gitsign** binary from the OpenShift cluster to your workstation.

a. Login to the OpenShift web console. From the home page, click the ? icon, click **Command**

a. Login to the OpenShift web console. From the home page, click the **?** icon, click **Command line tools**, go to the **gitsign** download section, and click the link for your platform.

b. Open a terminal on your workstation, decompress the .gz file, and set the execute bit:

Example

```
$ gunzip gitsign-amd64.gz
$ chmod +x gitsign-amd64
```

c. Move and rename the binary to a location within your **$PATH** environment:

Example

```
$ sudo mv gitsign-amd64 /usr/local/bin/gitsign
```

2. Switch to the RHTAS project:

Syntax

```
oc project PROJECT_NAME
```

Example

```
$ oc project trusted-artifact-signer
```

NOTE

Use the project name for the RHTAS installation.

3. Configure your shell environment for doing commit signing and verifying:

Example

```
$ export TUF_URL=$(oc get tuf -o jsonpath='{.items[0].status.url}' -n trusted-artifact-signer)
$ export OIDC_ISSUER_URL=https://$(oc get route keycloak -n keycloak-system | tail -n 1 |
awk '{print $2}')/auth/realms/trusted-artifact-signer
$ export COSIGN_FULCIO_URL=$(oc get fulcio -o jsonpath='{.items[0].status.url}' -n
trusted-artifact-signer)
$ export COSIGN_REKOR_URL=$(oc get rekor -o jsonpath='{.items[0].status.url}' -n trusted-
artifact-signer)
$ export COSIGN_MIRROR=$TUF_URL
$ export COSIGN_ROOT=$TUF_URL/root.json
$ export COSIGN_OIDC_CLIENT_ID="trusted-artifact-signer"
$ export COSIGN_OIDC_ISSUER=$OIDC_ISSUER_URL
$ export COSIGN_CERTIFICATE_OIDC_ISSUER=$OIDC_ISSUER_URL
$ export COSIGN_YES="true"
$ export SIGSTORE_FULCIO_URL=$COSIGN_FULCIO_URL
$ export SIGSTORE_OIDC_ISSUER=$COSIGN_OIDC_ISSUER
$ export SIGSTORE_REKOR_URL=$COSIGN_REKOR_URL
$ export REKOR_REKOR_SERVER=$COSIGN_REKOR_URL
```

4. Configure the local repository configuration to sign your commits by using the RHTAS service:

Example

```
$ git config --local commit.gpgsign true
$ git config --local tag.gpgsign true
$ git config --local gpg.x509.program gitsign
$ git config --local gpg.format x509
$ git config --local gitsign.fulcio $SIGSTORE_FULCIO_URL
$ git config --local gitsign.rekor $SIGSTORE_REKOR_URL
$ git config --local gitsign.issuer $SIGSTORE_OIDC_ISSUER
$ git config --local gitsign.clientID trusted-artifact-signer
```

5. Make a commit to the local repository:

   Example

   ```
   $ git commit --allow-empty -S -m “Test of a signed commit”
   ```

   A web browser opens allowing you to sign the commit with an email address.

6. Initialize The Update Framework (TUF) system:

   Example

   ```
   $ cosign initialize
   ```

7. Verify the commit:

   Syntax

   ```
   gitsign verify --certificate-identity=SIGNING_EMAIL --certificate-oidc-
   issuer=$SIGSTORE_OIDC_ISSUER HEAD
   ```

   Example

   ```
   $ gitsign verify --certificate-identity=jdoe@redhat.com --certificate-oidc-
   issuer=$SIGSTORE_OIDC_ISSUER HEAD
   ```

Additional resources

- Installing Red Hat Trusted Artifact Signer on OpenShift .
- Customizing Red Hat Trusted Application Pipeline .
- See the Signing and verifying containers by using Cosign from the command-line interface section in the RHTAS Deployment Guide for details on signing and verifying container images.
- The Update Framework home page.

2.3. VERIFYING SIGNATURES ON CONTAINER IMAGES WITH ENTERPRISE CONTRACT

Enterprise Contract (EC) is a tool for maintaining the security of software supply chains, and you can use it to define and enforce policies for container images. You can use the **ec** binary to verify the attestation and signature of container images that use Red Hat's Trusted Artifact Signer (RHTAS) signing framework.

Prerequisites

- A RHTAS installation on Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.
- A workstation with the **oc**, **cosign**, and **podman** binaries installed.
- Access to the OpenShift web console.

Procedure

1. Download the **ec** binary from the OpenShift cluster.
    a. Log in to the OpenShift web console. From the home page, click the **?** icon, click **Command line tools**, go to the **ec** download section, then click the link for your platform.
    b. Open a terminal on your workstation, decompress the binary .gz file, and set the execute bit:

    Example

    ```
    $ gunzip ec-amd64.gz
    $ chmod +x ec-amd64
    ```

    c. Move and rename the binary to a location within your **$PATH** environment:

    Example

    ```
    $ sudo mv ec-amd64 /usr/local/bin/ec
    ```

2. Switch to the RHTAS project:

    Syntax

    ```
    oc project PROJECT_NAME
    ```

    Example

    ```
    $ oc project trusted-artifact-signer
    ```

    NOTE

    Use the project name for the RHTAS installation.

3. Configure your shell environment for doing container image signing and verifying.

    Example

    ```
    $ export TUF_URL=$(oc get tuf -o jsonpath='{.items[0].status.url}' -n trusted-artifact-signer)
    $ export OIDC_ISSUER_URL=https://$(oc get route keycloak -n keycloak-system | tail -n 1 |
    awk '{print $2}')/auth/realms/trusted-artifact-signer
    ```

```
$ export COSIGN_FULCIO_URL=$(oc get fulcio -o jsonpath='{.items[0].status.url}' -n
trusted-artifact-signer)
$ export COSIGN_REKOR_URL=$(oc get rekor -o jsonpath='{.items[0].status.url}' -n trusted-
artifact-signer)
$ export COSIGN_MIRROR=$TUF_URL
$ export COSIGN_ROOT=$TUF_URL/root.json
$ export COSIGN_OIDC_CLIENT_ID="trusted-artifact-signer"
$ export COSIGN_OIDC_ISSUER=$OIDC_ISSUER_URL
$ export COSIGN_CERTIFICATE_OIDC_ISSUER=$OIDC_ISSUER_URL
$ export COSIGN_YES="true"
$ export SIGSTORE_FULCIO_URL=$COSIGN_FULCIO_URL
$ export SIGSTORE_OIDC_ISSUER=$COSIGN_OIDC_ISSUER
$ export SIGSTORE_REKOR_URL=$COSIGN_REKOR_URL
$ export REKOR_REKOR_SERVER=$COSIGN_REKOR_URL
```

4. Initialize The Update Framework (TUF) system:

   Example

   ```
   $ cosign initialize
   ```

5. Sign a test container image.

   a. Create an empty container image:

      Example

      ```
      $ echo "FROM scratch" > ./tmp.Dockerfile
      $ podman build . -f ./tmp.Dockerfile -t ttl.sh/rhtas/test-image:1h
      ```

   b. Push the empty container image to the **ttl.sh** ephemeral registry:

      Example

      ```
      $ podman push ttl.sh/rhtas/test-image:1h
      ```

   c. Sign the container image:

      Syntax

      ```
      cosign sign -y IMAGE_NAME:TAG
      ```

      Example

      ```
      $ cosign sign -y ttl.sh/rhtas/test-image:1h
      ```

      A web browser opens allowing you to sign the container image with an email address.

   d. Remove the temporary Docker file:

      Example

      ```
      $ rm ./tmp.Dockerfile
      ```

6. Create a **predicate.json** file:

   Example

   ```
   {
     "builder": {
       "id": "https://localhost/dummy-id"
     },
     "buildType": "https://example.com/tekton-pipeline",
     "invocation": {},
     "buildConfig": {},
     "metadata": {
       "completeness": {
         "parameters": false,
         "environment": false,
         "materials": false
       },
       "reproducible": false
     },
     "materials": []
   }
   ```

   Refer to the SLSA provenance predicate specifications for more information on the purpose, and schema layout.

7. Associate the **predicate.json** file with the container image:

   Syntax

   ```
   cosign attest -y --predicate ./predicate.json --type slsaprovenance IMAGE_NAME:TAG
   ```

   Example

   ```
   $ cosign attest -y --predicate ./predicate.json --type slsaprovenance ttl.sh/rhtas/test-image:1h
   ```

8. Verify that the container image has at least one attestation and signature:

   Syntax

   ```
   cosign tree IMAGE_NAME:TAG
   ```

   Example

   ```
   $ cosign tree ttl.sh/rhtas/test-image:1h

     Supply Chain Security Related artifacts for an image: ttl.sh/rhtas/test-
   image@sha256:7de5fa822a9d1e507c36565ee0cf50c08faa64505461c844a3ce3944d23efa35

    └──  Attestations for an image tag: ttl.sh/rhtas/test-image:sha256-
   7de5fa822a9d1e507c36565ee0cf50c08faa64505461c844a3ce3944d23efa35.att
      └──
   sha256:40d94d96a6d3ab3d94b429881e1b470ae9a3cac55a3ec874051bdecd9da06c2e
    └──  Signatures for an image tag: ttl.sh/rhtas/test-image:sha256-
   ```

```
7de5fa822a9d1e507c36565ee0cf50c08faa64505461c844a3ce3944d23efa35.sig
   └──
sha256:f32171250715d4538aec33adc40fac2343f5092631d4fc2457e2116a489387b7
```

9. Verify the container image by using Enterprise Contact:

Syntax

```
ec validate image --image IMAGE_NAME:TAG --certificate-identity-regexp
'SIGNER_EMAIL_ADDR' --certificate-oidc-issuer-regexp 'keycloak-keycloak-system' --output
yaml --show-successes
```

Example

```
$ ec validate image --image ttl.sh/rhtas/test-image:1h --certificate-identity-regexp
'jdoe@example.com' --certificate-oidc-issuer-regexp 'keycloak-keycloak-system' --output
yaml --show-successes

success: true
successes:
  - metadata:
      code: builtin.attestation.signature_check
    msg: Pass
  - metadata:
      code: builtin.attestation.syntax_check
    msg: Pass
  - metadata:
      code: builtin.image.signature_check
    msg: Pass
ec-version: v0.1.2427-499ef12
effective-time: "2024-01-21T19:57:51.338191Z"
key: ""
policy: {}
success: true
```

Enterprise Contract generates a pass-fail report with details on any security violations. When you add the **--info** flag, the report includes more details and possible solutions for any violations found.

Additional resources

- Installing Red Hat Trusted Artifact Signer on OpenShift .
- Managing compliance with Enterprise Contract.
- See the Enterprise Contract website for more information.

CHAPTER 3. CONFIGURE ADDITIONAL OPENID CONNECT PROVIDERS

3.1. CONFIGURING GOOGLE AS AN OPENID CONNECT PROVIDER FOR TRUSTED ARTIFACT SIGNER

You can use Google OAuth 2.0 as your OpenID Connect (OIDC) provider for Red Hat's Trusted Artifact Signer (RHTAS) service. You can decide to configure Google OAuth during the deployment of RHTAS, or at a later time.

IMPORTANT

You can define several different OIDC providers in the same configuration.

Prerequisites

- Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.
- Access to the OpenShift web console with the **cluster-admin** role.
- A workstation with the **oc**, and **podman** binaries installed.
- From the Google Cloud Console, create an OAuth client ID with the following settings:
  - Set the application type to “Web Application”.
  - Authorized redirect URIs must include: http://localhost/auth/callback .

Procedure

1. Open a terminal on your workstation, and log in to OpenShift:

   Syntax

   ```
   oc login --token=TOKEN --server=SERVER_URL_AND_PORT
   ```

   Example

   ```
   $ oc login --token=sha256~ZvFDBvoIYAbVECixS4-WmkN4RfnNd8Neh3y1WuiFPXC --
   server=https://example.com:6443
   ```

   NOTE

   You can find your login token and URL for use on the command line from the OpenShift web console. Log in to the OpenShift web console. Click your user name, and click **Copy login command**. Offer your user name and password again, if asked, and click **Display Token** to view the command.

2. Update the RHTAS configuration.

   a. Open for editing the **Securesign** resource:

Syntax

```
oc edit Securesign NAME -n NAMESPACE
```

Example

```
$ oc edit Securesign securesign-sample -n trusted-artifact-signer
```

> **NOTE**
>
> You must use the project name created for the RHTAS installation as the namespace.

b. Under the **OIDCIssuers** section, add a new subsection with your Google client identifier, issuer's URL, and set the **Type** value to **email**:

Syntax

```
...
OIDCIssuers:
  - Issuer: "https://accounts.google.com"
    IssuerURL: "https://accounts.google.com"
    ClientID: "CLIENT_ID"
    Type: email
...
```

Add you Google client identifier to the **ClientID** field.

c. Save your changes, and quit the editor. After a few seconds the operator automatically reconfigures the RHTAS software stack.

3. Change the OIDC issuer, and client id environment variables to use Google:

Example

```
$ export OIDC_ISSUER_URL=https://accounts.google.com
$ export COSIGN_OIDC_CLIENT_ID="314919563931-
35zke44ouf2oiztjg7v8o8c2ge9usnd1.apps.googleexample.com"
```

4. Copy and paste your secret from the Google Console to a plain text file:

Syntax

```
echo SECRET > my-google-client-secret
```

5. If you already have the RHTAS service running, you can verify the updated configuration by signing a test container image.

a. Create an empty container image:

Example

```
$ echo "FROM scratch" > ./tmp.Dockerfile
$ podman build . -f ./tmp.Dockerfile -t ttl.sh/rhtas/test-image:1h
```

b. Push the empty container image to the **ttl.sh** ephemeral registry:

Example

```
$ podman push ttl.sh/rhtas/test-image:1h
```

c. Remove the temporary Docker file:

Example

```
$ rm ./tmp.Dockerfile
```

d. Sign the container image:

Syntax

```
cosign sign -y --oidc-client-secret-file=SECRET_FILE IMAGE_NAME:TAG
```

Example

```
$ cosign sign -y --oidc-client-secret-file=my-google-client-secret ttl.sh/rhtas/test-image:1h
```

A web browser opens allowing you to sign the container image with an email address.

Additional resources

- Creating Google OAuth credentials.

3.2. CONFIGURING RED HAT SSO AS AN OPENID CONNECT PROVIDER FOR TRUSTED ARTIFACT SIGNER

You can use Red Hat Single Sign-On (SSO) as your OpenID Connect provider for Red Hat's Trusted Artifact Signer (RHTAS) service. This gives you a Keycloak authentication environment for applications and secure services.

Prerequisites

- Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.
- Access to the OpenShift web console with the **cluster-admin** role.
- Have 1 GB of container storage available for the Keycloak PostgreSQL database.
- A workstation with the **oc** binary installed.

Procedure

1. Log in to the OpenShift web console with a user that has the **cluster-admin** role.

2. Create a new project to deploy the Keycloak service.
   a. From the **Administrator** perspective, expand **Home** from the navigation menu, and click **Projects**.
   b. Click the **Create Project** button.
   c. The new **project name** is **keycloak-system**, and click the **Create** button.
3. Expand **Operators** from the navigation menu, and click **OperatorHub**.
4. In the search field, type **sso**, and click the **Red Hat Single Sign-on** tile.
5. Click the **Install** button to show the operator details.
6. If not already set, select **keycloak-system** from the **Installed Namespace** drop-down menu.
7. Click **Install** on the *Install Operator* page, and wait for the installation to finish.
8. After the installation finishes, click **View Operator**.
9. From your workstation terminal, log in to the OpenShift cluster:

   Syntax

   ```
   oc login --token=TOKEN --server=SERVER_URL_AND_PORT
   ```

   Example

   ```
   $ oc login --token=sha256~ZvFDBvoIYAbVECixS4-WmkN4RfnNd8Neh3y1WuiFPXC --
   server=https://example.com:6443
   ```

   NOTE

   You can find your login token and URL to use on the command line from the OpenShift web console. Log in to the OpenShift web console. Click your user name, and click **Copy login command**. Offer your user name and password again, if asked, and click **Display Token** to view the command.

10. Switch to the Keycloak project:

    Example

    ```
    $ oc project keycloak-system
    ```

11. Create a Keycloak instance:

    Example

    ```
    $ cat <<EOF | oc apply -f -
    apiVersion: keycloak.org/v1alpha1
    kind: Keycloak
    metadata:
      labels:
    ```

```
    app: sso
  name: keycloak
spec:
  externalAccess:
    enabled: true
  instances: 1
  keycloakDeploymentSpec:
    imagePullPolicy: Always
  postgresDeploymentSpec:
    imagePullPolicy: Always
EOF
```

12. Create a Keycloak realm:

Example

```
$ cat <<EOF | oc apply -f -
apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
  labels:
    app: sso
  name: trusted-artifact-signer
spec:
  instanceSelector:
    matchLabels:
      app: sso
  realm:
    displayName: Red-Hat-Trusted-Artifact-Signer
    enabled: true
    id: trusted-artifact-signer
    realm: trusted-artifact-signer
    sslRequired: none
EOF
```

13. Create a Keycloak client:

Example

```
$ cat <<EOF | oc apply -f -
apiVersion: keycloak.org/v1alpha1
kind: KeycloakClient
metadata:
  labels:
    app: sso
  name: trusted-artifact-signer
spec:
  client:
    attributes:
      request.object.signature.alg: RS256
      user.info.response.signature.alg: RS256
    clientAuthenticatorType: client-secret
    clientId: trusted-artifact-signer
    defaultClientScopes:
    - profile
    - email
    description: Client for Red Hat Trusted Artifact Signer authentication
    directAccessGrantsEnabled: true
    implicitFlowEnabled: false
    name: trusted-artifact-signer
    protocol: openid-connect
    protocolMappers:
    - config:
        claim.name: email
        id.token.claim: "true"
        jsonType.label: String
        user.attribute: email
        userinfo.token.claim: "true"
      name: email
      protocol: openid-connect
      protocolMapper: oidc-usermodel-property-mapper
    - config:
        claim.name: email-verified
        id.token.claim: "true"
        user.attribute: emailVerified
        userinfo.token.claim: "true"
      name: email-verified
      protocol: openid-connect
      protocolMapper: oidc-usermodel-property-mapper
    - config:
        claim.name: aud
        claim.value: trusted-artifact-signer
        id.token.claim: "true"
        access.token.claim: "true"
        userinfo.token.claim: "true"
      name: audience
      protocol: openid-connect
      protocolMapper: oidc-hardcoded-claim-mapper
    publicClient: true
    standardFlowEnabled: true
    redirectUris:
    - "*"
  realmSelector:
    matchLabels:
      app: sso
EOF
```

14. Create a Keycloak user:

Example

```
$ cat <<EOF | oc apply -f -
apiVersion: keycloak.org/v1alpha1
kind: KeycloakUser
metadata:
  labels:
    app: sso
  name: jdoe
spec:
  realmSelector:
    matchLabels:
      app: sso
  user:
    email: jdoe@redhat.com
    enabled: true
    emailVerified: true
    credentials:
      - type: "password"
        value: "secure"
    firstName: Jane
    lastName: Doe
    username: jdoe
EOF
```

Set a user name, the user’s email address, and a password or reference a secret object.

15. Go back to the OpenShift web console, click the **All instances** tab to watch and wait until the Keycloak system initializes successfully.

Additional resources

- Installing Red Hat Trusted Artifact Signer on OpenShift .

3.3. CONFIGURING AMAZON STS AS AN OPENID CONNECT PROVIDER FOR TRUSTED ARTIFACT SIGNER

You can use Amazon’s Security Token Service (STS) as your OpenID Connect (OIDC) provider for Red Hat’s Trusted Artifact Signer (RHTAS) service. You can decide to configure Amazon STS during the deployment of RHTAS, or at a later time.

IMPORTANT

You can define several different OIDC providers in the same configuration.

Prerequisites

- A RHTAS installation on Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.
- Access to the OpenShift web console with the **cluster-admin** role.
- A workstation with the **oc**, **podman**, and **aws** binaries installed.
- Enable managed Amazon Web Service (AWS) Resources for OpenShift environments.
- A created Amazon Identity and Access Management (IAM) user with full permissions. This allows access to run IAM operations.
    - Created access keys for this user.

Procedure

1. Open a terminal on your workstation, and log in to OpenShift:

   Syntax

```
oc login --token=TOKEN --server=SERVER_URL_AND_PORT
```

Example

```
$ oc login --token=sha256~ZvFDBvoIYAbVECixS4-WmkN4RfnNd8Neh3y1WuiFPXC --
server=https://example.com:6443
```

> **NOTE**
>
> You can find your login token and URL for use on the command line from the OpenShift web console. Log in to the OpenShift web console. Click your user name, and click **Copy login command**. Offer your user name and password again, if asked, and click **Display Token** to view the command.

2. Find the AWS OIDC provider URL:

   Example

   ```
   $ oc get authentication cluster -o jsonpath='{.spec.serviceAccountIssuer}'
   ```

3. Update RHTAS the configuration.

   a. Open for editing the **Securesign** resource:

      Syntax

      ```
      oc edit Securesign NAME -n NAMESPACE
      ```

      Example

      ```
      $ oc edit Securesign securesign-sample -n trusted-artifact-signer
      ```

      > **NOTE**
      >
      > You must use the project name created for the RHTAS installation as the namespace.

   b. Under the **OIDCIssuers** section, add a new subsection with your AWS STS client identifier, issuer's URL, and set the **Type** value to **kubernetes**:

      Example

      ```
      ...
      OIDCIssuers:
        - Issuer: "https://example.s3.us-east-1.aws.com/47bd6cg0vs5nn01mue83fbof94dj4m9c"
          IssuerURL: "https://example.s3.us-east-
      1.aws.com/47bd6cg0vs5nn01mue83fbof94dj4m9c"
          ClientID: "trusted-artifact-signer"
          Type: kubernetes
      ...
      ```

   c. Save your changes, and quit the editor. After a few seconds the operator automatically reconfigures the RHTAS software stack.

4. Configure the AWS command-line tool by entering your access key, secret key, default region, and output format:

   Example

```
$ aws configure
```

5. Set the following environment variables:

   Example

```
$ export account_id=$(aws sts get-caller-identity --query "Account" --output text)
$ export oidc_provider="$(oc get authentication cluster -o
jsonpath='{.spec.serviceAccountIssuer}' | cut -d '/' -f3-)"
$ export role_name=rhtas-sts
$ export namespace=rhtas-sts
$ export service_account=cosign-sts
```

6. Create a Trust Policy that gets associated with newly created IAM roles:

   Example

```
$ cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::${account_id}:oidc-provider/${oidc_provider}"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "${oidc_provider}:aud": "trusted-artifact-signer"
        }
      }
    }
  ]
}
EOF
```

7. Create a new IAM role for the RHTAS service by using the trust policy:

   Example

```
$ aws iam create-role --role-name rhtas-sts --assume-role-policy-document file://trust-
relationship.json --description "Red Hat Trusted Artifact Signer STS Role"
```

8. On the OpenShift cluster with STS enabled, create a new project namespace:

Syntax

```
oc new-project NAMESPACE
```

Example

```
$ oc new-project rhtas-sts
```

9. Create a service account for assuming an IAM role, and running a workload within the OpenShift project namespace.

    a. Create the service account manifest:

    Example

    ```
    $ cat >service_account.yaml <<EOF
    apiVersion: v1
    kind: ServiceAccount
    metadata:
      name: $service_account
      namespace: $namespace
      annotations:
        eks.amazonaws.com/role-arn: "arn:aws:iam::${account_id}:role/${role_name}"
        # optional: Defaults to "sts.amazonaws.com" if not set
        eks.amazonaws.com/audience: "trusted-artifact-signer"
        # optional: When "true", adds AWS_STS_REGIONAL_ENDPOINTS env var to
    containers
        eks.amazonaws.com/sts-regional-endpoints: "true"
        # optional: Defaults to 86400 for expirationSeconds if not set
        eks.amazonaws.com/token-expiration: "86400"
    EOF
    ```

    b. Apply the service account manifest to OpenShift:

    Example

    ```
    $ oc apply -f service_account.yaml
    ```

10. Create a new deployment workload for signing container images within a image registry.

    a. Create the deployment manifest:

    Example

    ```
    $ cat >deployment.yaml <<EOF
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: cosign-sts
      namespace: ${namespace}
    spec:
      selector:
        matchLabels:
          app: cosign-sts
    ```

```
  template:
    metadata:
      labels:
        app: cosign-sts
    spec:
      securityContext:
        runAsNonRoot: true
      serviceAccountName: cosign-sts
      containers:
      - args:
        - -c
        - env; cosign initialize --mirror=\$COSIGN_MIRROR --root=\$COSIGN_ROOT;
while true; do sleep 86400; done
        command:
        - /bin/sh
        name: cosign
        image: registry.redhat.io/rhtas-tech-preview/cosign-
rhel9@sha256:f4c2cec3fc1e24bbe094b511f6fe2fe3c6fa972da0edacaf6ac5672f06253a3e

        pullPolicy: IfNotPresent
        env:
        - name: AWS_ROLE_SESSION_NAME
          value: signer-identity-session
        - name: AWS_REGION
          value: us-east-1
        - name: OPENSHIFT_APPS_SUBDOMAIN
          value: $(oc get cm -n openshift-config-managed  console-public -o go-template="{{
.data.consoleURL }}" | sed 's@https://@@; s/^[^.]*\.//')
        - name: OIDC_AUTHENTICATION_REALM
          value: "trusted-artifact-signer"
        - name: COSIGN_FULCIO_URL
          value: $(oc get fulcio -o jsonpath='{.items[0].status.url}' -n trusted-artifact-signer)
        - name: COSIGN_OIDC_ISSUER
          value: $(oc get authentication cluster -o jsonpath='{.spec.serviceAccountIssuer}')
        - name: COSIGN_CERTIFICATE_OIDC_ISSUER
          value: $(oc get authentication cluster -o jsonpath='{.spec.serviceAccountIssuer}')
        - name: COSIGN_REKOR_URL
          value: $(oc get rekor -o jsonpath='{.items[0].status.url}' -n trusted-artifact-signer)
        - name: COSIGN_MIRROR
          value: $(oc get tuf -o jsonpath='{.items[0].status.url}' -n trusted-artifact-signer)
        - name: COSIGN_ROOT
          value: "$(oc get tuf -o jsonpath='{.items[0].status.url}' -n trusted-artifact-
signer)/root.json"
        - name: COSIGN_YES
          value: "true"
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext:
        runAsNonRoot: true
      serviceAccount: ${service_account}
      serviceAccountName: ${service_account}
      terminationGracePeriodSeconds: 30
EOF
```

b. Apply the deployment manifest to OpenShift:

Example

```
$ oc apply -f deployment.yaml
```

11. Create a test container image to sign.

a. Create an empty container image:

Example

```
$ echo "FROM scratch" > ./tmp.Dockerfile
$ podman build . -f ./tmp.Dockerfile -t ttl.sh/rhtas/test-image:1h
```

b. Push the empty container image to the **ttl.sh** ephemeral registry:

Example

```
$ podman push ttl.sh/rhtas/test-image:1h
```

c. Remove the temporary Docker file:

Example

```
$ rm ./tmp.Dockerfile
```

12. Validate the configuration by signing and verifying the test container image.

a. Open a remote shell session within a running pod:

Syntax

```
oc rsh -n NAMESPACE deployment/cosign-sts env IMAGE=IMAGE_NAME:TAG /bin/sh
```

Example

```
$ oc rsh -n rhtas-sts deployment/cosign-sts env IMAGE=ttl.sh/rhtas/test-image:1h /bin/sh
```

b. Sign the container image:

Example

```
$ cosign sign -y --identity-token=$(cat $AWS_WEB_IDENTITY_TOKEN_FILE)
ttl.sh/rhtas/test-image:1h
```

c. Verify the signed container image:

Example

```
$ cosign verify --certificate-identity=https://kubernetes.io/namespaces/$(cat
/var/run/secrets/kubernetes.io/serviceaccount/namespace)/serviceaccounts/cosign-sts --
certificate-oidc-issuer=$COSIGN_CERTIFICATE_OIDC_ISSUER ttl.sh/rhtas/test-
image:1h
```

3.4. CONFIGURING GITHUB AS AN OPENID CONNECT PROVIDER FOR TRUSTED ARTIFACT SIGNER

You can use GitHub OAuth 2.0 when federating it with Red Hat's Single Sign-On (SSO) service as an OpenID Connect (OIDC) provider for the Red Hat Trusted Artifact Signer (RHTAS) service. This procedure guides you on integrating GitHub OAuth with an existing Red Hat SSO deployment on OpenShift.

IMPORTANT

You can define several different OIDC providers in the same configuration.

Prerequisites

- A RHTAS installation on Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.
- A running Red Hat SSO instance.
- A workstation with the **oc** binary installed.
- Create a GitHub OAuth app , and after registering the application, make note of the client identifier and secret values.

  IMPORTANT

  When registering your new GitHub OAuth app, you must specify the **Homepage URL**, and the **Authorization callback URL** Enter placeholder values for both of these fields, for example, **https://localhost:8080**. Later in this procedure you will modify your GitHub OAuth app with the intended values for these fields.

Procedure

1. Open a terminal on your workstation, and log in to OpenShift:

   #### Syntax

   ```
   oc login --token=TOKEN --server=SERVER_URL_AND_PORT
   ```

   #### Example

   ```
   $ oc login --token=sha256~ZvFDBvoIYAbVECixS4-WmkN4RfnNd8Neh3y1WuiFPXC --
   server=https://example.com:6443
   ```

NOTE

You can find your login token and URL for use on the command line from the OpenShift web console. Log in to the OpenShift web console. Click your user name, and click **Copy login command**. Offer your user name and password again, if asked, and click **Display Token** to view the command.

2. Log in to the Red Hat SSO console.
    a. Find the Red Hat SSO console URL from the command line:

    Example

    ```
    $ oc get routes -n keycloak-system keycloak -o jsonpath='https://{.spec.host}'
    ```

    b. Copy and paste the Red Hat SSO console URL into your web browser.
    c. Click **Administration Console**.
    d. Retrieve the **admin** password from the command line:

    Example

    ```
    $ oc get secret/credential-keycloak -n keycloak-system -o jsonpath='{
    .data.ADMIN_PASSWORD }' | base64 -d
    ```

    Copy the output from this command.

    e. From your web browser, log in as the **admin** user, and paste the password in the corresponding field. Click the **Sign In** button.
3. Select your realm from the dropdown on the navigation menu.
4. Add the GitHub identity provider.
    a. From the navigational menu, click **Identity Providers**.
    b. From the **Add provider...** drop-down menu, select **GitHub**.
    c. Add your GitHub OAuth client identifier to the **Client ID** field.
    d. Add your GitHub OAuth client secret to the **Client Secret** field.
    e. Turn **on** the **Trust Email** option.
    f. Click the **Save** button.
5. Add the identity provider mapper to the newly created identity provider.
    a. Click the **Mapper** tab.
    b. Click the **Create** button.
    c. Give a **Name** to the new mapper.
    d. Change the **Mapper Type** to **Hardcoded Attribute**.

e. Set the **User Attribute** field to **emailVerified**.

f. Set the **User Attribute Value** field to **true**.

g. Click the **Save** button.

6. From the *GitHub Identity Provider Settings* page, copy the **Redirect URI** value and paste it to your GitHub OAuth app **Authorization Callback URL** field. Also, paste this same value into the **Homepage URL** field, but remove the **broker/github/endpoint** part of the URL string.

7. Click **Update Application**. You can now sign commits, and containers by using GitHub as your OIDC provider.

8. When signing artifacts, a web browser opens and prompts you to sign in to your Red Hat SSO account. Click the **GitHub** button to sign in with your credentials.

9. Click the **Authorize** button to enable GitHub user details to be accessible by Red Hat SSO.

Additional resources

- Installing Red Hat Trusted Artifact Signer on OpenShift .
- Using cosign for signing and verifying .
- Using gitsign for signing and verifying .

CHAPTER 4. CONFIGURE AN ALTERNATIVE DATABASE FOR TRUSTED ARTIFACT SIGNER

You can replace the Red Hat Trusted Artifact Signer (RHTAS) default database for Trillian with an externally managed MariaDB database instance. The database instance can be a cloud-hosted database provider, such as Amazon's Relational Database Service (RDS), or your own database deployment in OpenShift.

4.1. PREREQUISITES

- Red Hat OpenShift Container Platform version 4.13, 4.14, or 4.15.

4.2. CONFIGURING AMAZON RDS FOR TRUSTED ARTIFACT SIGNER

With this procedure, you can replace Red Hat's Trusted Artifact Signer (RHTAS) default database for Trillian with a MariaDB instance managed by Amazon's Relational Database Service (RDS).

IMPORTANT

Red Hat recommends using a highly available MariaDB database for production workloads.

Prerequisites

- An Amazon Web Service (AWS) account with access to the Amazon RDS console.
- Access to the OpenShift web console with the **cluster-admin** role.
- A workstation with the **oc**, **curl**, and the **mysql** binaries installed.
- Command-line access with privileges to create a database and populate the MariaDB instance.

Procedure

1. Open the Amazon RDS console, and create a new MariaDB instance.
   a. Wait for the MariaDB instance to be deployed, and is available.
2. From your workstation, log in to the new database by providing the regional endpoint, the port, and the user credentials:

   Syntax

   ```
   mysql -h REGIONAL_ENDPOINT -P 3306 -u USER_NAME -p
   ```

   Example

   ```
   $ mysql -h exampledb.1234.us-east-1.rds.amazonaws.com -P 3306 -u admin -p
   ```

3. Create a new database named trillian:

   Example

```
create database trillian;
```

4. Switch to the newly created database:

   Example

   ```
   use trillian;
   ```

5. Create a new database user named **trillian**, and set a *PASSWORD* for the newly created user:

   Syntax

   ```
   CREATE USER trillian@'%' IDENTIFIED BY 'PASSWORD';
   GRANT ALL PRIVILEGES ON trillian.* TO 'trillian'@'%';
   FLUSH PRIVILEGES;
   ```

6. Disconnect from the database:

   Example

   ```
   EXIT
   ```

7. Download the database configuration file:

   Example

   ```
   $ curl -o dbconfig.sql
   https://raw.githubusercontent.com/securesign/trillian/main/storage/mysql/schema/storage.sql
   ```

8. Apply the database configuration to the new database:

   Syntax

   ```
   mysql -h FQDN_or_SERVICE_ADDR -P 3306 -u USER_NAME -p PASSWORD -D
   DB_NAME < PATH_TO_CONFIG_FILE
   ```

   Example

   ```
   $ mysql -h rhtasdb.example.com -P 3306 -u trillian -p mypassword123 -D trillian <
   dbconfig.sql
   ```

9. Open a terminal on your workstation, and log in to OpenShift:

   Syntax

   ```
   oc login --token=TOKEN --server=SERVER_URL_AND_PORT
   ```

   Example

   ```
   $ oc login --token=sha256~ZvFDBvoIYAbVECixS4-WmkN4RfnNd8Neh3y1WuiFPXC --
   server=https://example.com:6443
   ```

NOTE

You can find your login token and URL for use on the command line from the OpenShift web console. Log in to the OpenShift web console. Click your user name, and click **Copy login command**. Offer your user name and password again, if asked, and click **Display Token** to view the command.

10. Create a new Secret containing the credentials for the Trillian database within the MariaDB instance which was created previously:

    Syntax

    ```
    oc create secret generic OBJECT_NAME \
    --from-literal=mysql-database=trillian \
    --from-literal=mysql-host=FQDN_or_SERVICE_ADDR \
    --from-literal=mysql-password=PASSWORD \
    --from-literal=mysql-port=3306 \
    --from-literal=mysql-root-password=PASSWORD \
    --from-literal=mysql-user=USER_NAME
    ```

    Example

    ```
    $ oc create secret generic trillian-mysql \
    --from-literal=mysql-database=trillian \
    --from-literal=mysql-host=mariadb.trusted-artifact-signer.svc.cluster.local \
    --from-literal=mysql-password=mypassword123 \
    --from-literal=mysql-port=3306 \
    --from-literal=mysql-root-password=myrootpassword123 \
    --from-literal=mysql-user=trillian
    ```

    You can use an OpenShift internal service name for the MariaDB instance.

11. You can now deploy the Trusted Artifact Signer service to use this database. If you were following the Trusted Artifact Signer installation procedure, then you can proceed to the next step.

Additional resources

- Creating new secret objects in OpenShift .

4.3. CONFIGURING A DATABASE IN OPENSHIFT FOR TRUSTED ARTIFACT SIGNER

With this procedure, you can replace Red Hat's Trusted Artifact Signer (RHTAS) default database for Trillian with a MariaDB instance managed by Amazon's Relational Database Service (RDS).

IMPORTANT

Red Hat recommends using a highly available MariaDB database for production workloads.

Prerequisites

- Permissions to create an OpenShift project, and deploy a database instance from the OpenShift samples catalog.
- Access to the OpenShift web console with the **cluster-admin** role.
- A workstation with the **oc**, **curl**, and the **mysql** binaries installed.
- Command-line access with privileges to create a database and populate the MariaDB instance.

Procedure

1. Log in to the OpenShift web console where you are deploying the RHTAS service:
2. Change to the **Developer** perspective.
3. Select the **trusted-artifact-signer** project, if the project already exists, else create a new project for the database:
   a. To create a new project, click the drop-down project menu, and click the **Create Project** button.
   b. Name the new project **trusted-artifact-signer**, and click the **Create** button.
4. On the *Developer Catalog* card, click **Database**.
5. Select **MariaDB**, and click the **Instantiate Template** button.

   IMPORTANT

   Do not select **MariaDB (Ephemeral)**.

6. On the *Instantiate Template* page, configure the following fields:
   a. In the **MariaDB Database Name** field, enter **trillian**.
   b. In the **Volume Capacity** field, enter **5Gi**.
   c. Click the **Create** button.
7. Begin a remote shell session:
   a. On the *Topology* page, selecting the MariaDB pod brings up a side panel, click the **Resources** tab.
   b. Under the *Pods* section, click on the MariaDB pod name.
   c. Click the **Terminal** tab to start a remote shell session to the MariaDB pod.
8. In the remote shell session, verify that you can connect to the Trillian database:

   Example

   ```
   $ mysql -u $MYSQL_USER -p$MYSQL_PASSWORD -D$MYSQL_DATABASE
   ```

NOTE

Credentials are stored in a secret object with the service name (**mariadb**), and contains the name of the database, and user name, along with the database root password. Make a note of these credentials as they will be used later on when creating the database secret object.

9. Disconnect from the database:

   Example

   ```
   EXIT
   ```

10. Download the database configuration file:

    Example

    ```
    $ curl -o dbconfig.sql
    https://raw.githubusercontent.com/securesign/trillian/main/storage/mysql/schema/storage.sql
    ```

11. Apply the database configuration to the new database:

    Syntax

    ```
    mysql -h FQDN_or_SERVICE_ADDR -P 3306 -u USER_NAME -p PASSWORD -D
    DB_NAME < PATH_TO_CONFIG_FILE
    ```

    Example

    ```
    $ mysql -h rhtasdb.example.com -P 3306 -u trillian -p mypassword123 -D trillian <
    dbconfig.sql
    ```

12. Open a terminal on your workstation, and log in to OpenShift:

    Syntax

    ```
    oc login --token=TOKEN --server=SERVER_URL_AND_PORT
    ```

    Example

    ```
    $ oc login --token=sha256~ZvFDBvoIYAbVECixS4-WmkN4RfnNd8Neh3y1WuiFPXC --
    server=https://example.com:6443
    ```

NOTE

You can find your login token and URL for use on the command line from the OpenShift web console. Log in to the OpenShift web console. Click your user name, and click **Copy login command**. Offer your user name and password again, if asked, and click **Display Token** to view the command.

13. Create a new Secret containing the credentials for the Trillian database within the MariaDB instance which was created previously:

    Syntax

    ```
    oc create secret generic OBJECT_NAME \
    --from-literal=mysql-database=trillian \
    --from-literal=mysql-host=FQDN_or_SERVICE_ADDR \
    --from-literal=mysql-password=PASSWORD \
    --from-literal=mysql-port=3306 \
    --from-literal=mysql-root-password=PASSWORD \
    --from-literal=mysql-user=USER_NAME
    ```

    Example

    ```
    $ oc create secret generic trillian-mysql \
    --from-literal=mysql-database=trillian \
    --from-literal=mysql-host=mariadb.trusted-artifact-signer.svc.cluster.local \
    --from-literal=mysql-password=mypassword123 \
    --from-literal=mysql-port=3306 \
    --from-literal=mysql-root-password=myrootpassword123 \
    --from-literal=mysql-user=trillian
    ```

    You can use an OpenShift internal service name for the MariaDB instance.

14. You can now deploy the Trusted Artifact Signer service to use this database. If you were following the Trusted Artifact Signer installation procedure, then you can proceed to the next step.

Additional resources

- Creating new secret objects in OpenShift .

APPENDIX A. TRUSTED ARTIFACT SIGNER COMPONENTS AND VERSION NUMBERS

The following tables list Red Hat's Trusted Artifact Signer (RHTAS) software components and their corresponding version numbers for the 1.0.2 release.

Table A.1. Binaries

Component	Version
cosign	2.2.3
gitsign	0.8.1
rekor	1.3.6
ec	0.2.1

Table A.2. Trillian

Component	Version
Log Server	1.6.0
Log Signer	1.6.0
Database	1.6.0
Redis	1.6.0

Table A.3. Rekor

Component	Version
Server	1.3.6
Backfill Redis	1.3.6
Rekor Search UI	1.3.6

Table A.4. Fulcio

Component	Version
Server	1.4.5

Table A.5. Certificate Transparency

Component	Version
Certificate Transparency Go	1.1.8

Table A.6. Scaffolding

Component	Version
The Update Framework (TUF) Server	0.6.16
createctconfig	0.6.16
ctlog-managectroots	0.6.16
trillian-createtree	0.6.16
trillian-createdb	0.6.16
fulcio-createcerts	0.6.16

Additional resources

- For more information about Trillian, see the project page on GitHub.
- For more information about Sigstore’s Rekor, see the project page on GitHub.
- For more information about Sigstore’s Fulcio, see the project page on GitHub.
- For more information about Sigstore’s Timestamp Authority, see the project page on GitHub.
- For more information about TUF, see their home page hosted by the Linux Foundation.